CPS3230 – Fundamentals of Software Testing

ASSIGNMENT PART 1 OF 3 – MARKET ALERT UM

Robert Fenech Adami

B.Sc. (Hons.) Information Technology (Software Development)

November 2022

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as "the unacknowledged use, as one's own work, of work of another person, whether or not such work has been published" (Regulations Governing Conduct at Examinations, 1997, Regulation 1 (viii), University of Malta).

I / We*, the undersigned, declare that the [assignment / Assigned Practical Task report / Final Year Project report] submitted is my / our* work, except where acknowledged and referenced.

I / We* understand that the penalties for making a false declaration may include, but are not limited to, loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

* Delete as appropriate.

(N.B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

Robert Fenech Adami
_____   _____
Student Name   Signature

_____   _____
Student Name   Signature

CPS3230   Assignment Part 1 of 3 – Market Alert UM
_____   _____
Course Code   Title of work submitted
13/11/2022
_____
Date

Task 1

Github repository: https://github.com/robbyfa/marketAlertTask1

Google Drive:
https://drive.google.com/drive/folders/1wZVTt368XZxm4r4mASsn_p7OxxzBagyF?usp=sharing

The SUT implemented consists of an 'Alert' object which includes all its relative details. An 'AlertCatalogue' class is implemented. This class contains a list of all alert objects created, together with relative methods to gather data through web scraping, publish alerts to marketAlertUM and, also, delete all alerts from the site, among other methods. A utilities folder which consists of several interfaces is also used. The architecture for the system under test can be seen in figure 1 below. All data was gathered from:
https://www.ultimate.com.mt/product-category/tv-audio/audio/headphones/
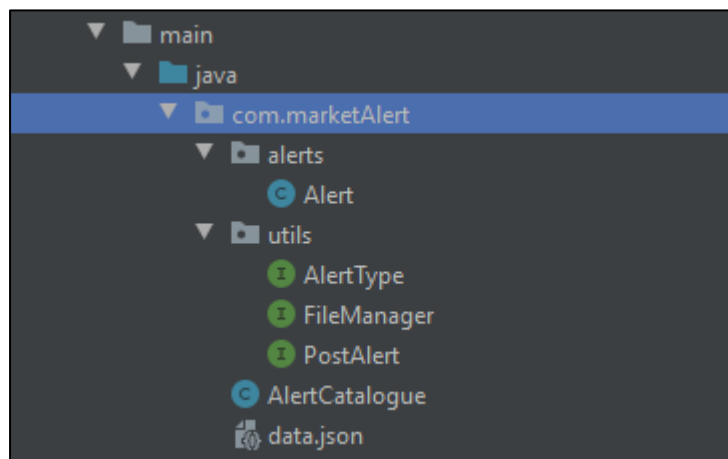
[Fig.1]

Several tests have been implemented for each aspect of the system using a test-driven approach, thus obtaining 100% test coverage, as seen in Figure 2 below. Each test was implemented using a 'setup, verify, exercise, and verify architecture; Furthermore, these tests also make use of spies in order to verify the number of alerts being saved to the alerts list and to verify the number of alerts being published to marketAlertUM via the REST API [fig. 3] [fig. 4]. The tests also make use of Mockito to assign the alert type to each alert being published.

[Fig.2]

```
FileManagerSpy.java ×
1      package marketAlert.spies;
2
3
4      import marketAlert.alerts.Alert;
5      import marketAlert.utils.FileManager;
6
7      public class FileManagerSpy implements FileManager {
8
9          public int numCallsToSaveAlertToFile = 0;
10
11         public void saveProductToFile(Alert alert) { numCallsToSaveAlertToFile++; }
14
15     }
16
```

[Fig.3]

It is assumed that all data being gathered via the Chrome driver web scraper fall under the electronics category when publishing alerts via the REST API. Furthermore, it is assumed that all data, for 5 alerts, is being stored at one go, rather than adding and publishing one alert at a time. Further to this, the method used to call the web scraper simply implements a loop to gather alert data until the maximum limit is reached.

Another measure taken to ensure proper web scraping is the use of Gson and FileWriter dependency and library. These are used to store all data retrieved from the automated web scraper to a JSON file, as seen below in figure 5.

```
PostSpy.java ×
1      package marketAlert.spies;
2
3
4      import marketAlert.utils.PostAlert;
5      💡
6      public class PostSpy implements PostAlert {
7
8          public int alertsPosted = 0;
9
10         public void postItem() { alertsPosted++; }
13         }
14
15
```

[Fig.4]

```
data.json ×
1  {
2      "alertType": 6,
3      "heading": "JBL Endurance Peak II Blue",
4      "description": "Waterproof True Wireless In-Ear Sport Headphones",
5      "url": "https://www.ultimate.com.mt/product/jbl-endurance-peak-ii-blue/",
6      "imageUrl": "https://www.ultimate.com.mt/wp-content/uploads/2020/11/JBL_ENDURANCE_PEAK-II_Product-Image_Hero-2_Blue-200x200.png",
7      "postedBy": "ff557502-1ba4-4578-b094-2efdd4375b1d",
8      "priceInCents": 9900
9  }
```

[Fig. 5]

The below figure shows the success status of all tests, with 22 of 22 passes.
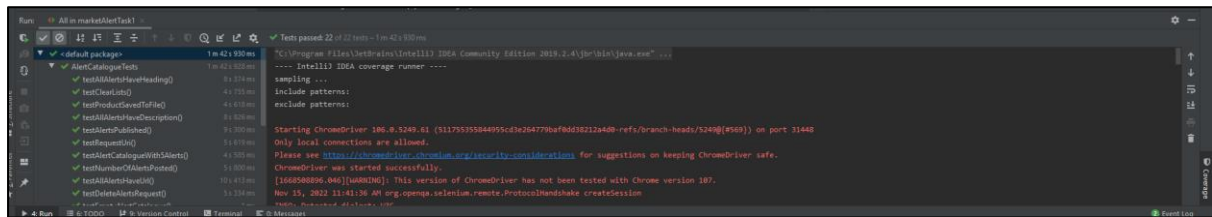
AlertCatalogueTests:

Test Name	Scope
testEmptyAlertCatalogue	Assure alerts list is empty
testAlertCatalogueWith5Alerts	Assure web scraper gathers data for 5 alerts
testProductSavedToFile	Assure 5 alerts are saved to file
testNumberOfAlertsPosted	Assure number of alerts being posted
testHttpRequest	Assure successful use of API endpoint
testRequestUri	Assure proper HTTP request URI
testDeleteAlertsRequest	Assure successful use of API 'Delete' endpoint
testClearLists	Assure alerts list is emptied
testAlertsPublished	Count number of alerts being posted to marketalertum
testAllAlertsHave	Assure all alerts posted have al required properties

AlertTests:

Test Name	Scope
testHeading	Assure heading variable is assigned
testDescription	Assure description variable is assigned
testUrl	Assure url variable is assigned
testImageUrl	Assure imageUrl variable is assigned
testPostedBy	Assure postedBy variable is assigned
testAlertType	Assure alertType variable is

	assigned
testPriceInCents	Assure priceInCents variable is assigned

| Name	Scope
FileManagerSpy	Used to store number of alerts saved
PostSpy	Used to store number of alerts being posted
AlertType (Mockito)	Used to set alert type for alerts

Task 2

Github repository: https://github.com/robbyfa/marketAlertTask2

Google Drive:
https://drive.google.com/drive/folders/1wZVTt368XZxm4r4mASsn_p7OxxzBagyF?usp=sharing

With regards to the SUT to be tested, all methods are declared within one 'MarketAlert' class. All feature files have been stored in the 'resources' folder in the test suite and all respective steps and runners have been stored in separate folders, i.e., the steps and runner for test1 are stored in the same package, while the steps and runner for test 2 are stored together in a different package. This was done to allow different features to have identical steps and thus avoid repetition of similar methods in the SUT. The project architecture can be seen below in Figure 6.
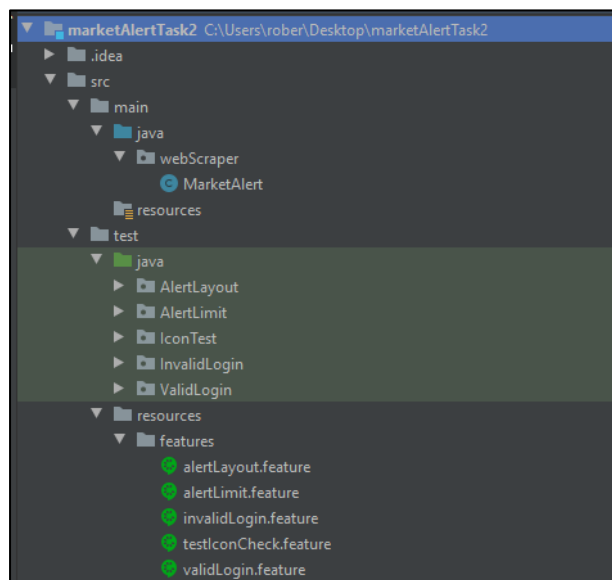
[Fig. 6]

Upon running tests, as seen in figure 7 below, all 5 tests are successful.
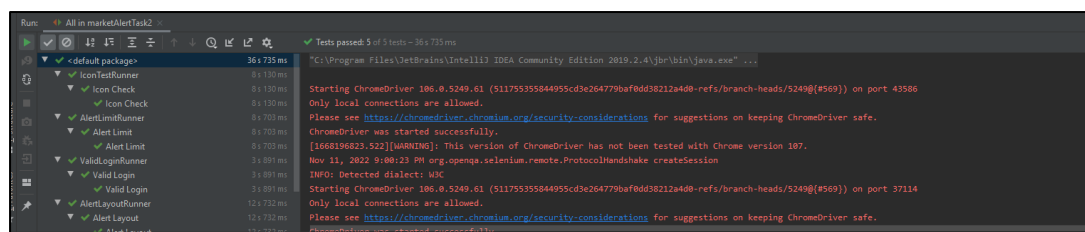
[Fig. 7]

With regards to the testability of 'marketalertum.com', it is suggested that an alert counter element is implemented in the 'My Alerts' page to allow quicker verification when verifying the number of alerts published.

Valid Login Steps:

Label	Step	Purpose
Given	I am a user of marketalertum	Creates marketAlert instance
When	I login using valid credentials	Call valid login method
Then	I should see my alerts	Verify url

Invalid Login steps

Label	Step	Purpose
Given	I am a user of marketalertum	Creates marketAlert instance
When	I login using valid credentials	Call valid login method
Then	I should see the login screen again	Verify url

Alert Layout Steps:

Label	Step	Purpose
Given	I am a user of marketalertum	Creates marketAlert instance
When	I view a list of alerts	Call valid login method
Then / And	Each alert should contain a …	Find number of element

Alert Limit Steps:

Label	Step	Purpose
Given	I am an administrator of the website and I upload more than {int} alerts	Creates marketAlert instance and call addAlerts method.
Given	I am a user of marketalertum	Call marketAlert instance
When	I view a list of alerts	Call valid login method
Then	I should see {int} alerts	Count alerts and verify number